

0808.65559

PATENT APPLICATION

8-#3
15-01
JC971 U.S. PTO
09/862654
05/22/01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re U.S. Patent Application)
Applicant: Nigel Peter Topham and)
Adrian Wise)
Serial No.)
Filed: May 22, 2001)
For: PROCESSING HAVING)
COMPRESSED ...)
Art Unit:)

I hereby certify that this paper is being deposited with the United States Postal Service as EXPRESS MAIL in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231, on May 22, 2001.

Express Label No.: EL 846164444 US

Signature: Daniel Anon

CLAIM FOR PRIORITY

Assistant Commissioner for Patents
Washington, DC 20231

Sir:

Applicant claims foreign priority benefits under 35 U.S.C. § 119 on the basis of the foreign application identified below:

European Patent Application No. 0012839.7, filed May 25, 2000.

A certified copy of the priority document is enclosed.

Respectfully submitted,

GREER, BURNS & CRAIN, LTD.

By



Patrick G. Burns
Reg. No. 29,367

May 22, 2001
300 South Wacker Drive
Suite 2500
Chicago, IL 60606
(312) 360-0080
Customer Number: 24978

THIS PAGE BLANK (USPTO)



INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

JC971 U.S. PTO

09/862654



I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed

Dated

20 APR 2001

THIS PAGE BLANK (USPTO)



**Statement of inventorship and of
right to grant of a patent**

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

25 MAY 2000

1. Your reference HL74355/000/PMH

2. Patent application number **0012839.7**
(if you know it)

3. Full name of the or of each applicant Siroyan Limited

4. Title of the invention

Processors having compressed instructions and methods of compressing instructions for processors

5. State how the applicant(s) derived the right
from the inventor(s) to be granted a patent

By virtue of the inventor's contract of employment with the applicant

6. How many, if any, additional Patent Forms
7/77 are attached to this form? None
(See note (c))

7. I/We believe that the person(s) named over the page (and on
any extra copies of this form) is/are the inventor(s) of the invention
which the above patent application relates to.

Signature

Date

Handwritten signature

25 May 2000

8. Name and daytime telephone number of
person to contact in the United Kingdom Mr P M Hitching
[020] 7420 0500

Notes

- a) If you need help to fill in this form or you have any questions, please contact the patent office on 0645 500505.
- b) Write your answers in capital letters using black ink or you may type them.
- c) If there are more than three inventors, please write the names and addresses of the other inventors on the back of another Patents Form 7/77 and attach it to this form.
- d) When an application does not declare any priority, or declares priority from an earlier UK application, you must provide enough copies of this form so that the patent office can send one to each inventor who is not an applicant.
- e) Once you have filled in the form you must remember to sign and date it.

Enter the full names, addresses and postcodes of the inventors in the boxes and underline the surnames

Nigel Peter Topham

6 Carolina Place
Finchampstead
Wokingham RG40 4PQ
England

7830797002
~~7830797001~~

Patents ADP number (if you know it)

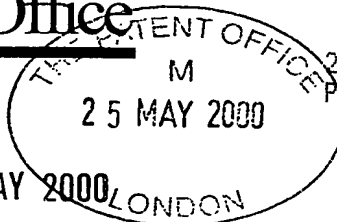
Patents ADP number (if you know it)

Patents ADP number (if you know it)

Reminder:
Have you signed this form?

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)



The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference HL74355/000/PMH

2. Patent application number
(The Patent Office will fill in this part)

0012839.7

3. Full name, address and postcode of the or of each applicant (underline all surnames)

Siroyan Limited

~~Wyvel's Court~~

Swallowfield

~~Reading RG7 1WY~~

12-15 HANGER GREEN WAVE

LONDON

W5 3AY

Patents ADP number (if you know it)

If the applicant is a corporate body, give the country/state of its incorporation

England

7630763001

7830763003

MS 1/23/1/10
28/11/00

4. Title of the invention
Processors having compressed instructions and methods of compressing instructions for processors

5. Full name of your agent (if you have one)

Haseltine Lake & Co.

"Address for service" in the United Kingdom to which all correspondence should be sent (including the postcode)

Imperial House
15-19 Kingsway
London WC2B 6UD

Patents ADP number (if you know it)

34001

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and (if you know it) the or each application number

Country

Priority application number
(if you know it)

Date of filing
(day/month/year)

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application

Date of filing
(day/month/year)

8. Is a statement of inventorship and of right to a grant of patent required in support of this request? (Answer "Yes" if:

Yes

a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form

Description	34	/
Claim(s)	9	/
Abstract	1	/ WS-
Drawing(s)	13	+13 /

10. If you are also filing any of the following,

Priority documents

Translations of priority documents

Statement of inventorship and right to a grant of patent (*Patents Form 7/77*)

2 /

Request for preliminary examination and search (*Patents Form 9/77*)

1 /

Request for substantive examination (*Patents Form 10/77*)

Any other documents
(*please specify*)

11.

I/We request the grant of a patent on the basis of this application

Signature *Harold L. P.* Date 25 May 2000

12. Name and daytime telephone number of person to contact in the United Kingdom

Mr P M Hitching

[020] 7420 0500

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

- If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.
- Write your answers in capital letters using black ink or you may type them.
- If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.
- If you have answered "Yes" Patents Form 7/77 will need to be filed.
- Once you have filled in the form you must remember to sign and date it.

DUPLICATE¹

-1-

PROCESSORS HAVING COMPRESSED INSTRUCTIONS AND
METHODS OF COMPRESSING INSTRUCTIONS FOR PROCESSORS

5 The present invention relates to processors having compressed instructions. In particular, but not exclusively, the present invention relates to very long instruction word (VLIW) processors having compressed instructions. The present invention also relates to methods of comprising instructions for processors.

10 A VLIW instruction schedule (program) may contain a significant number of "no operation" (NOP) instructions which are there simply to pad out empty slots in the overall instruction schedule. As it is wasteful to store such NOPs explicitly in a schedule or
15 program memory used for storing the instruction schedule, it is desirable to provide a mechanism for storing the VLIW instructions in the schedule memory in a compressed form.

20 Fig. 1(A) of the accompanying drawings shows an example original (non-compressed) VLIW instruction schedule made up of three VLIW packets P0, P1 and P2. Each packet is made up of two instructions. In this example, therefore, the processor which is to execute the instruction schedule must have first and second
25 execution units, the first instruction of each packet (instruction 1) being executed by the first execution unit in parallel with the execution of the second instruction (instruction 2) of that packet by the second execution unit.

30 In the Fig. 1(A) example, half of the slots in the schedule contain NOP instructions (slots 1, 2 and 4).

35 Fig. 1(B) shows how the instruction schedule of Fig. 1(A) would be stored in its original non-compressed form in the schedule memory. In Fig. 1(B) the instructions appear as a sequential scan from left to right and from top to bottom of the VLIW instruction

schedule of Fig. 1(A).

Fig. 1(C) shows how the Fig. 1(A) schedule can be stored in the schedule memory in compressed (or compacted) form. The first word of the compressed
5 schedule contains a bit vector, referred to hereinafter as a "decompression key". The decompression key has a plurality of bits corresponding respectively to the instructions in the non-compressed schedule (Fig. 1(B)). If a particular bit in the key is a 0 this
10 denotes that the instruction corresponding to that bit is a NOP instruction. If the bit is a 1 its corresponding instruction is a useful (non-NOP) instruction. In this way, all NOP instructions can be eliminated in the compressed version of the schedule.

15 Such a compression mechanism is highly valuable in an embedded processing environment (in which the processor is embedded in a system such as in a mobile communication device) where high code or instruction density is of critical importance because of the
20 limited resources of the system, for example in terms of available program memory. However, such compression complicates the task of executing instructions in parallel. For example, when a VLIW instruction schedule contains two instructions which could in
25 principle be executed in parallel but which are separated by a number of NOP instructions, the processor would have to search linearly through the compressed version of the schedule to identify instructions that could be executed in parallel. Most
30 importantly, after compression, concurrency between one instruction and other instructions can no longer be determined simply by observing the position of that one instruction relative to those other instructions as they are stored in the schedule memory. In general,
35 one of the primary advantages of VLIW processing (over more complex schemes for issuing instructions in

parallel such as superscalar processing) is that in a (non-compressed) VLIW instruction schedule it is possible to determine when instructions are independent of one another (and hence can be executed concurrently) by observing the relative positions of instructions in the schedule. Accordingly, it is desirable to facilitate determination of independence even in a situation in which the instruction schedule is stored in the schedule memory in compressed form.

When a VLIW instruction schedule is stored in compressed form in the schedule memory the compressed packets must of course be decompressed before they can be supplied to the execution units for execution of the instructions contained therein. The decompression is desirably performed "on-the-fly", i.e. during actual execution of the instruction schedule. To make such on-the-fly decompression possible, the decompression must be performed with low computational complexity and involve a comparatively simple hardware implementation so that the cost, in terms of lost execution time, arising from the decompression process is small.

According to a first aspect of the present invention there is provided a processor, for executing instructions of a program stored in compressed form in a program memory, comprising: a program counter for identifying a position in the said program memory; an instruction cache, having a plurality of cache blocks, each for storing one or more instructions of the said program in decompressed form; cache loading means, including decompression means, operable to perform a cache loading operation in which one or more compressed-form instructions are read from the said position in the program memory identified by the program counter and are decompressed and stored in one of the said cache blocks of the instruction cache; a cache pointer for identifying a position in the said

instruction cache of an instruction to be fetched for execution; instruction fetching means for fetching an instruction to be executed from the position identified by the cache pointer and operable, when a cache miss occurs because the instruction to be fetched is not present in the instruction cache, to cause the cache loading means to perform such a cache loading operation; and updating means for updating the program counter and cache pointer in response to the fetching of instructions so as to ensure that the said position identified by the said program counter is maintained consistently at the position in the said program memory at which the instruction to be fetched from the instruction cache is stored in compressed form.

According to a second aspect of the present invention there is provided a method of compressing a program to be executed by a processor in which compressed-form instructions stored in a program memory are decompressed and cached in an instruction cache prior to being issued, the method including the steps of: converting a sequence of original instructions of the program into a corresponding sequence of such compressed-form instructions; assigning such original instructions imaginary addresses according to the said sequence thereof, the assigned imaginary addresses being imaginary addresses at which the instructions are to be considered to exist when held in decompressed form in the said instruction cache of the processor; and storing, in the said program memory, the compressed-form instructions together with imaginary address information specifying the said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can assign the specified imaginary addresses to the decompressed instructions.

According to a third aspect of the present invention there is provided a computer program which, when run on a computer, causes the computer to carry out a method of compressing a processor program to be executed by a processor, the processor being operable to decompress compressed-form instructions stored in a program memory and to cache the decompressed instructions in an instruction cache prior to issuing them, the computer program including: a converting portion for converting a sequence of original instructions of the processor program into a corresponding sequence of such compressed-form instructions; an assigning portion for assigning such original instructions imaginary addresses according to the said sequence thereof, the assigned imaginary addresses being imaginary address at which the instructions are to be considered to exist when held in decompressed form in the said instruction cache of the processor; and a storing portion for storing, in the said program memory, the compressed-form instructions together with imaginary address information specifying the said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the processor can assign the specified imaginary addresses to the decompressed instructions.

Reference will now be made, by way of example, to the accompanying drawings, in which:

Figs. 1(A), 1(B) and 1(C) show explanatory diagrams for illustrating compression of a VLIW instruction schedule;

Fig. 2 shows parts of a processor embodying the present invention;

Fig. 3 shows parts of an instruction issuing unit in a first embodiment of the present invention;

Fig. 4 is an explanatory diagram for illustrating

compression of a VLIW instruction schedule in the Fig. 3 embodiment;

Fig. 5 is a diagram showing the internal organisation of parts of an instruction cache in Fig. 3;

Fig. 6 shows parts of the Fig. 3 instruction cache in more detail;

Fig. 7 is a diagram showing an example format of a cache tag in the Fig. 3 instruction cache;

Fig. 8 shows parts of an instruction issuing unit in a second embodiment of the present invention;

Fig. 9 is an explanatory diagram for illustrating a difficulty in branching in imaginary memory space;

Fig. 10 shows a VLIW instruction schedule prior to compression in a worked example for illustrating operation of the Fig. 8 embodiment;

Fig. 11 is a diagram showing how the VLIW instruction schedule of Fig. 10 is stored in compressed form in a schedule memory; and

Figs. 12 to 20 are respective diagrams for illustrating an instruction cache state and an updating unit state at different stages in the Fig. 10 worked example; and

Fig. 21 shows a flowchart for use in explaining a method of compressing instructions according to another aspect of the present invention.

Fig. 1 shows parts of a processor embodying the present invention. In this example, the processor is a very long instruction word (VLIW) processor. The processor 1 includes an instruction issuing unit 10, a schedule storage unit 12, respective first, second and third execution units 14, 16 and 18, and a register file 20. The instruction issuing unit 10 has three issue slots IS1, IS2 and IS3 connected respectively to the first, second and third execution units 14, 16 and 18. A first bus 22 connects all three execution units

14, 16 and 18 to the register file 20. A second bus 24 connects the first and second units 14 and 16 (but not the third execution unit 18 in this embodiment) to a memory 26 which, in this example, is an external random access memory (RAM) device. The memory 26 could alternatively be a RAM internal to the processor 1.

Incidentally, although Fig. 1 shows shared buses 22 and 24 connecting the execution units to the register file 20 and memory 26, it will be appreciated that alternatively each execution unit could have its own independent connection to the register file and memory.

The processor 1 performs a series of processing cycles. In each processing cycle the instruction issuing unit 10 can issue one instruction at each of the issue slots IS1 to IS3. The instructions are issued according to a VLIW instruction schedule (described below) stored in the schedule storage unit 12.

The instructions issued by the instructing issuing unit 10 at the different issue slots are executed by the corresponding execution units 14, 16 and 18. In this embodiment each of the execution units can execute more than one instruction at the same time, so that execution of a new instruction can be initiated prior to completion of execution of a previous instruction issued to the execution unit concerned.

To execute instructions, each execution unit 14, 16 and 18 has access to the register file 20 via the first bus 22. Values held in registers contained in the register file 20 can therefore be read and written by the execution units 14, 16 and 18. Also, the first and second execution units 14 and 16 have access via the second bus 24 to the external memory 26 so as to enable values stored in memory locations of the external memory 26 to be read and written as well. The

third execution unit 18 does not have access to the external memory 26 and so can only manipulate values contained in the register file 20 in this embodiment.

Fig. 3 is a block diagram showing parts of the instruction issuing unit 10 of the Fig. 2 processor in a first embodiment of the present invention.

In this embodiment, the instruction issuing unit 10 includes an instruction cache 40, a cache loading unit 42 having a decompression section 44, an instruction fetching unit 46, an updating unit 48 and an instruction register 54. The updating unit 48 includes three registers in this embodiment: a program counter register (PC register) 50, a compressed instruction counter register (CC register) 51 and a cache pointer register (VPC register) 52.

The cache loading unit 42 is connected to the schedule storage unit 12 for receiving therefrom compressed-form VLIW instructions VCS. The cache loading unit 42 is also connected to the instruction fetching unit 46 for receiving therefrom a control signal LOAD, and is also connected to the PC register 50 for receiving the PC value held therein.

The instruction cache 40 is connected to the cache loading unit 42 for receiving therefrom decompressed instructions DI, as well as a compressed instruction count value (CC) associated with the decompressed instructions DI. The instruction cache 40 is also connected to the instruction fetching unit 46 for receiving therefrom a control signal FETCH and for outputting thereto a control signal MISS. The instruction cache 40 is further connected to the VPC register 52 in the updating unit 48 for receiving therefrom the VPC value held therein.

The instruction register 54 is connected to the instruction cache 40 for receiving therefrom a selected processor packet PP. The instruction register 54 in

this embodiment has a width of eight instructions, providing eight issue slots IS1 to IS8. Each issue slot is connected to an individually-corresponding execution unit (not shown).

5 The instruction fetching unit 46 is connected to the updating unit 48 for applying thereto a control signal UPDATE, as well as the above-mentioned LOAD signal.

10 The VPC register 52 is the updating unit 48 is also connected to the cache loading unit for receiving therefrom an extracted VPC value EVPC associated with the decompressed instructions DI. The CC register 51 in the updating unit 48 is connected to the instruction cache 40 for receiving therefrom an accessed cache
15 block instruction count vale ACC.

Operation of the units shown in Fig. 3 will now be described with reference to Figs. 4 to 7.

20 The Fig. 2 processor may operate selectively in two modes: a scalar mode and a VLIW mode. In scalar mode the processor executes instructions from a particular instruction set (which may or may not be distinct from the VLIW instruction set) but does not attempt to issue instructions in parallel at the issue slots IS1 to IS8. In VLIW mode, on the other hand, up
25 to 8 instructions are issuable in parallel per instruction cycle at the 8 issue slots IS1 to IS8, i.e. the full instruction issue width is exploited.

30 Scalar-mode instructions and VLIW-mode instructions are both stored together in the schedule storage unit 12, with the VLIW instructions being stored in a predetermined compressed form. The program counter (PC) value held in the PC register 50 is used to identify the position reached in the stored sequence of instructions in the schedule storage unit 12, both
35 in the scalar mode and in the VLIW mode. Operation in the scalar mode will not be considered in further

detail herein.

Fig. 4 shows a section VCS of VLIW instructions stored in compressed form in the schedule storage unit 12. This compressed form is essentially the same as that described hereinbefore with reference to Figs. 1(A) to 1(C), except that in the Fig. 4 section VCS the first word of the section VCS is used for storing an imaginary address value (VPC value), as will be explained in more detail hereinafter. The second word of the section VCS is used for storing the decompression key KEY needed for decompressing the instructions contained in the section VCS. The remaining words of the section VCS are used for storing any non-NOP instructions belonging to the section concerned. No NOP instructions are therefore stored explicitly in the section VCS.

When the processor attempts to execute the section VCS of compressed VLIW instructions the PC register 50 will initially point to the start of the section. In order to determine which instructions in the section VCS belong to the same processor packet (i.e. are instructions which must be issued simultaneously at the issue slots IS1 to IS8), and in which positions within that packet, the compressed section VCS must be decompressed. In the instruction issuing unit 10 of Fig. 3 the section VCS is compressed by the decompression section 44 and the resulting decompressed block of instructions DI is stored in the instruction cache 40. The block of decompressed instructions DI corresponding to the VLIW compression section VCS is therefore not actually stored in the schedule storage unit 12 even at execution time, and at execution time the decompressed instructions DI exist only in the instruction cache 40 in an "imaginary address space".

The mapping from the program address of the start of a compressed VLIW section VCS to its imaginary

address is created by an assembler/linker used to assemble/link the processor's program. The mapping information in the present embodiment is the VPC value shown in Fig. 4, stored in the first word of the compressed section VCS. Thus, as shown in Fig. 4, the PC register 50 points to the start of the compressed VLIW section VCS in the schedule storage unit (normal program memory) 12. The VPC value held in the first word of the section VCS is a pointer to the start of the decompressed block of instructions DI in imaginary memory (i.e. an entry point into the decompressed block DI).

In the present embodiment, as Fig. 4 shows, the decompressed block DI is made up of 32 words. This requires a 32-bit decompression key KEY. In a 32-bit processor, this means that the decompression key KEY occupies only one word in the compressed section VCS, corresponding to a space overhead for compression of 6.25% of the decompressed block size. When instruction schedules are dense (i.e. there are few NOPs) the overhead on the compressed code will approach 6.25%, which is an acceptable overhead. When schedules are sparse, however, the overhead on compressed code will be high in relation to the total amount of code, but the net saving in memory will be significant. If v is the fraction of instructions in a schedule that are not NOPs (i.e. v represents the code density) then the size S_{VCS} of a compressed section VCS will be $S_{VCS} = 2 + 32v$ words, and a net space saving will be achieved when $v < 93.8\%$, i.e. when any two or more instructions in a block of up to 32 instructions are NOPs.

Fig. 5 shows the internal organisation of the instruction cache 40 in this embodiment in more detail. As shown in Fig. 5 the instruction cache 40 is organised in rows and columns, with each row representing an individual processor packet PP and each

column representing the instructions within a processor packet. The instruction cache 40 is also sub-divided into a plurality (4 in this example) of cache blocks (CB0 to CB3). In this example, each cache block is made up of 32 words. As there are eight instructions in each processor packet, each cache block within the instruction cache 40 contains four processor packets.

The VPC value currently held in the VPC register 52 is used to identify the current processor packet being issued, i.e. loaded into the instruction register 54.

Fig. 6 shows the structure of the instruction cache 40 in this embodiment in more detail. The instruction cache 40 comprises an instruction memory unit 410, a tag unit 420, an index extraction unit 430, and a cache hit detection unit 440. The instruction memory unit 410 is used for storing the decompressed instructions and is organised into cache blocks as described already with reference to Fig. 5. Each cache block in the instruction memory unit 410 has an individually-associated cache tag CT held in the tag unit 420. An example of the format of each cache tag CT is shown in Fig. 7. In this example, the cache tag CT has three fields. The first field (V-field) is a single-bit field used to indicate the validity of the tag. When V=0 this indicates that the associated cache block does not contain valid data (instructions). When V=1 this indicates that the associated cache block does contain valid instructions.

The second field (CC) field of the cache tag CT is a five-bit field for storing a compressed instruction count value (CC) representing the number of non-NOP instructions in the associated cache block. The purpose of this field will be explained in more detail later in the present specification.

The third field (IBA field) is used to store an

imaginary block address (IBA) which effectively represents the address of the associated cache block in the imaginary address space described hereinbefore with reference to Fig. 4. The IBA field may be
5 approximately 32 bits in length.

Referring back to Fig. 6, when a cache block of the instruction memory unit 410 is to be accessed, the block is identified using the imaginary address value (VPC value) supplied from the VPC register 52 (Fig. 5).
10 In this embodiment, the cache 40 is a directly-mapped cache, and any particular address in the imaginary address space can only be mapped to a unique one of the cache blocks in the cache 40. The identification of the required cache block based on the received VPC
15 value is performed as follows.

Firstly, the index extraction unit 430 extracts an index value INDEX from the received VPC value. This index value INDEX is made up of a preselected group of successive bits (bit field) from within the received
20 VPC value. The number of bits in INDEX is i , where 2^i is the total number of cache blocks in the cache 40. The index value INDEX is used directly to address one cache tag CT from amongst the set of cache tags held in the tag unit 420.

25 The V and IBA fields of the addressed cache tag CT are output by the tag unit 420 to the cache hit detection unit 440.

When a match is found between the received VPC value and the IBA value held in the IBA field of the
30 cache tag, and the V field indicates the associated cache block contains valid instructions ($V=1$), the cache detection unit 440 determines that a cache "hit" has occurred. In this case, the higher-order address bits needed to address the associated cache block
35 within the instruction memory unit 410 are provided directly by the tag number TN of the matching cache

tag. In this way, the cache block is identified. To select an individual processor packet from within the identified block, lower-order address bits are required. For example, if each block contains four
5 processor packets (as in Fig. 5), two lower-order address bits are required. These lower-order address bits can be taken directly from the corresponding lower-order bits of the received VPC value.

If no cache tag having an IBA matching the
10 received VPC value is present in the tag unit 420, or if there is such a matching tag but the V field of that tag is 0, the cache hit detection unit 440 produces the MISS control signal to indicate a cache "miss" has occurred.

15 Incidentally, it will be appreciated that, because the VPC value is only used to identify processor packets, as opposed to individual instructions or even bytes within the processor packet, the least significant z bits of the VPC value (and also of each
20 IBA) are 0, where 2^z is the number of bytes in each processor packet. Accordingly, these least significant z bits are not implemented in the VPC register 52 or in the IBA field of each cache tag. Furthermore, as each IBA value is only used to identify an imaginary block
25 address, i.e. the imaginary address of the start of a cache block in which decompressed instructions DI are present, a further y least-significant bits of each IBA are also 0, where y is the number of processor packets in each cache block. These further y bits are also not
30 implemented in the IBA field of each cache tag.

Referring back to Fig. 3, when the FETCH control signal is applied to the instruction cache 40 by the instruction fetching unit 46, two outcomes are
35 possible: a cache hit or a cache miss. In the event of a cache hit, the current processor packet identified by the VPC value held in the VPC register 52 is loaded

directly into the instruction register 54, whereafter the UPDATE control signal is supplied by the instruction fetching unit 46 to the updating unit 48. In response to the UPDATE signal the VPC value held in the VPC register 52 is incremented to point to the next processor packet in the instruction cache. When a cache hit occurs in response to the FETCH signal, the value held in the CC field of the matching cache tag is loaded into the CC register 51, as well.

If a cache miss occurs in response to the FETCH signal, the cache hit detection unit 440 supplies the MISS signal to the instruction fetching unit 46. In this case, before the processor packet having the imaginary address specified by the current VPC value can be fetched into the instruction register 54, it is necessary for a cache loading operation to be performed to load a block of decompressed instructions, containing that processor packet, into an available one of the cache blocks of the instruction cache 40. Such a cache loading operation is initiated by the instruction fetching unit by applying the LOAD signal to the cache loading unit 42.

When a cache miss occurs, for reasons that will become apparent, the VPC value contained at the address in the schedule storage unit 12 pointed to by the PC value held in the PC register 50 will always match the current VPC value held in the VPC register 52. This means that loading of the required compressed-form VLIW code section VCS can be commenced immediately from that address.

Firstly, in the cache tag addressed by the INDEX value extracted by the index extraction unit 430 from the current VPC value, the V-field is set to 1 and the IBA field is loaded with the higher-order bits of the current VPC value held in the VPC register 52. In this way, the cache block associated with the addressed

cache tag is reserved for storing the decompressed instructions corresponding to the compressed section VCS pointed to by the PC register.

5 Secondly, an internal count value CC of the decompressed section 44 is set to 0.

Next, the decompression key KEY of the compressed-form VLIW code section VCS pointed to by the PC register is read from the schedule storage unit 12 at the storage location $PC+k$, where k is the number of
10 bytes in each word. The decompression key KEY is supplied to the decompression section 44.

The decompression section 44 examines each bit in turn of the decompression key KEY. If the bit is a 1, the cache loading unit 42 loads an instruction word of
15 the compressed section VCS from the schedule storage unit 12 at the address given by $PC+k(CC+1)$. The loaded instruction word is then stored in the reserved cache block at a position within the block corresponding to the examined bit. The internal count value CC is then
20 incremented by 1.

If the examined bit is 0, on the other hand, the decompression section 44 outputs a NOP instruction word, which is stored in the identified cache block at a position in that block corresponding to the examined
25 bit. The internal count value CC is not incremented in this case.

When all of the bits of the decompression key have been examined in this way, the internal count value CC finally reached is output by the cache loading unit 42
30 and stored in the CC field of the cache tag CT with which the reserved cache block is associated. This completes the cache loading operation.

After the cache loading operation is finished, the final step is to load the current processor packet PP
35 from the newly-loaded cache block into the instruction register 54. The CC field of the cache tag associated

with the newly-loaded cache block is output as the value ACC when that packet is loaded into the instruction register 54. This value ACC is stored in the CC register 51 of the updating unit.

5 It will be appreciated that in the Fig. 3 instruction issuing unit 10 the decompression of the VLIW code sections takes place "on-the-fly", that is, as the instructions are loaded into the cache. Such on-the-fly decompression is complicated by the fact
10 that the capacity of the instruction cache 40 is limited and that it is not uncommon for the processor to have to switch process, for example in response to the receipt of an interrupt. As a consequence, it is quite possible that between the issuance of two
15 successive processor packets belonging to the same cache block, the cache block concerned will have been evicted from the cache by another process bringing into the cache some of its own VLIW instructions. This means that in practice it is possible for any cache
20 access to result in a miss. Accordingly, at any instruction cycle, the processor must be capable of reloading the cache with the (decompressed) instructions belonging to the missing cache block. This presents a real problem in that the VPC value
25 (imaginary address of the decompressed instructions) held in the VPC register is of little use in locating the required compressed section VCS needed to obtain those decompressed instructions and there is no simple function that will translate from a VPC value to a PC
30 value at which the VLIW packet pointed to by VPC is located in compressed form.

 It is for this reason that in the Fig. 3 embodiment the PC and VPC values are always maintained consistent with one another by the updating unit 48.
35 In this way it is guaranteed whenever a cache miss occurs PC will be pointing to the start of the

compressed representation of the missing cache block pointed to by VPC. This consistency is ensured in the present embodiment by storing next-section locating information for use in locating the position in the program memory (i.e. a PC value) of the next compressed section following the compressed section whose corresponding cache block was accessed most recently to fetch an instruction.

In particular, the CC register 51 is updated, each time a cache block within the instruction cache 40 is accessed to fetch an instruction, with next-section locating information for use in locating the next compressed section after the compressed section corresponding to the accessed cache block. This next-section locating information in the present embodiment is the compressed instruction count value (CC value) for the compressed section corresponding to the most-recently-accessed cache block. This CC value represents the size of the compressed section corresponding to that most-recently-accessed cache block.

In the present embodiment, to enable the CC value for any valid cache block to be available immediately, the cache tag associated with each cache block holds in its CC field the CC value for the compressed section corresponding to the cache block concerned. The CC value to be stored in the CC field is generated by the decompression section 44 during the cache loading operation in which the compressed section is loaded into the cache. As the CC value for each valid cache block is generated at the time of cache loading and held in the CC field of the cache tag associated with that block, when any cache block is accessed to fetch an instruction, the CC value of that block's corresponding compressed section VCS can be obtained immediately by reading the CC field and storing the CC

value in the CC register 51. In this way, the CC register 51 will always contain the CC value of the compressed section corresponding to the most-recently-accessed cache block. Thus, when a cache miss occurs, the position in the program memory of the next compressed section following that compressed section can be obtained simply by setting $PC = PC + k(CC+2)$, where k is the number of bytes in each word. This makes reloading of any cache block possible at high speed when the block has been evicted between the fetching of a pair of successive packets belonging to that block.

It will be appreciated that the next-section locating information can take many other forms than a CC value. Each compressed section as stored could include the next-section locating information explicitly, for example a size value such as a CC value indicating the size of the section or even the direct address (PC value) of the start of the following compressed section. If the next-section locating information is held explicitly in the compressed section it is not necessary for the decompression section 44 to generate this information during the cache loading operation. However, in this case the compressed section will contain more words, reducing the memory savings available.

It is also not necessary to use the CC register 51 to hold the CC value of the compressed section corresponding to the most-recently-accessed cache block. As long as the most-recently-accessed cache block can always be identified in some way, the CC field of the cache tag associated with that block can be accessed "on demand" to provide the next-section locating information, although accessing the CC register will be faster.

Fig. 8 shows parts of an instruction issuing unit

110 according to a second embodiment of the present invention. The second embodiment is intended to enable on-the-fly decompression in a processor whose program is permitted to contain basic loops, as well as
5 straight-line code, in the VLIW portions of the program. A basic loop is a loop in which there are no other jumps, branches or separate routine calls.

In the Fig. 8 embodiment the instruction issuing unit 110 is constituted in basically the same way as
10 the instruction issuing unit 10 of the Fig. 3 embodiment and, in Fig. 8, elements of the instruction issuing unit 110 which are the same as, or correspond to, elements shown in Fig. 3 are denoted by the same reference numerals.

15 The Fig. 8 embodiment differs from the Fig. 3 embodiment in that the Fig. 8 embodiment has an updating unit 148 which, in addition to including the PC register 50, CC register 51 and VPC register 52, further includes five further registers 150 to 158.
20 These five registers are loop control registers provided specifically to improve the performance of basic loops of VLIW instructions.

In a basic loop, in general (i.e. other than when a process switch or other exception occurs) the next
25 block to be executed is either the next block beyond the current block or else it is a repetition of the first block of the loop. In the second embodiment, no other possibilities are permitted because of the extreme difficulty in executing an arbitrary relative
30 jump within the imaginary address space provided by the instruction cache 40 as illustrated in Fig. 9.

In Fig. 9, the left portion of the diagram shows an original portion UP of VLIW instructions prior to compression. In this example the portion UP is made up
35 of three basic blocks BB1 to BB3. A basic block is a sequence of instructions with a single entry point and

a single exit point. An entry point is any instruction that is the target of a branch, jump or call instruction. An exit point is any branch, jump or call instruction, or any instruction that is followed by an entry point. Thus, in Fig. 9 the first basic block BB1 has an exit point where it has a "branch if equal" instruction "BEQ label". The second basic block BB2 commences with the first instruction after that branch instruction and ends at the instruction immediately prior to the target instruction "label:" of the BEQ instruction, i.e. the entry point of the third basic block BB3.

After compression the compressed version CP of the program portion UP concerned is shown on the right on Fig. 9. The compressed version CP occupies three compressed sections VCS1, VCS2 and VCS3. Each such compressed section VCS will occupy one cache block in the instruction cache 40 after decompression. However, the boundaries between those cache blocks do not correspond to the boundaries between the basic blocks BB1 to BB3, as illustrated in Fig. 9. In the compressed form in which each cache block is stored in the program memory (schedule storage unit) there is no linear relationship between the storage address of the compressed cache block and the original basic blocks of VLIW code. The branch instruction at the end of the basic block BB1 must therefore specify its target address in such a way that the basic block BB3 can be found and that the offset of the target instruction within that block can be determined. This is highly problematic. For example, if the branch offset were specified as an offset within the imaginary memory space a linear scan of the compressed version CP would be needed to find the compressed section VCS containing that imaginary target address. Conversely, if the branch target were specified as an offset within the

(real) program memory space there would be no problem in identifying the first instruction at the target location, but the cache block in which that instruction occurs could not be identified. It might be considered
5 in this situation that branch instructions should specify both the real and imaginary addresses of the target location but in practice the run-time overhead involved in such a scheme would render it impractical.

In view of the difficulties associated with
10 arbitrary branching within imaginary address space the Fig. 8 embodiment is intended for use with a processor having a restricted programming model in which such arbitrary relative jumps are not permitted. Using such a restricted programming model, permitting only basic
15 loops, there is still the problem of how to branch back to the beginning of the basic loop. This problem is solved in the Fig. 8 embodiment using the loop control registers 150 to 158. Specifically, these registers are a loop PC register (LPC register) 150, a loop VPC
20 register (LVPC register) 152, an iteration counter register (IC register) 154, a loop size register (LSIZE register) 156, and a loop count register (LCNT register) 158.

Operation of the Fig. 8 embodiment is as follows.
25 The LVPC register 152 is used to store the imaginary address of the first processor packet of a basic loop of VLIW instructions. The LPC register 150 is used to store the address (virtual address) in the schedule storage unit 12 of the compressed section VCS
30 corresponding to the cache block pointed to by the LVPC register 152. The LPC and LVPC registers 150 and 152 are used together to refill the first cache block of a basic loop if it has been evicted from the instruction cache 40 between the initiations of any pair of
35 successive iterations of the loop.

Upon entry into a basic loop, the current values

of PC and VPC contained in the PC and VPC registers 50 and 52 are copied into the LPC and LVPC registers 150 and 152 respectively. The basic loop will be initiated by one or more predetermined instructions which will
5 cause the IC register 154 to be loaded with the number of iterations of the loop to be performed. The loop-initiating instruction(s) will also cause LSIZE register 156 to be loaded with the number of processor packets contained in the loop body. A copy of the
10 LSIZE value is also placed in the LCNT register 158.

During execution of the basic loop, when a processor packet is executed the LCNT register 158 is decremented by 1. When the LCNT value becomes 0 a new loop iteration is initiated.

15 When each new loop iteration is initiated the IC register 154 is decremented by 1. If it becomes 0 then all iterations of the loop have been completed. Otherwise, the LCNT register 158 is reloaded with the value held in the LSIZE register 156, the VPC register
20 52 is reloaded from the LVPC register 152, and the PC register 50 is reloaded from the LPC register 150.

When the basic loop completes, the VPC register 52 will be pointing to the first processor packet after the loop block. The processor status is then updated
25 to reflect the fact that the processor is no longer executing a basic loop, after which normal VLIW processing continues from the next processor packet.

Next, operation of the second embodiment of the present invention will be illustrated with a worked
30 example. In this worked example, a VLIW program portion to be executed is presented in Fig. 10 in its original form, i.e. prior to compression. It is assumed that the processor in this example is capable of issuing four instructions per instruction cycle. In
35 Fig. 10 a "-" indicates a NOP instruction.

As shown in Fig. 10, the example program portion

contains 20 useful (non-NOP) instructions I1 to I20, as well as a loop initiation instruction "loop 8, r1".

The instructions are allocated addresses in an imaginary address space from 1000 to 10bf (expressed in hexadecimal notation). It will also be assumed, in this example, that each cache block in the instruction cache 40 is made up of 64 bytes, so that the imaginary address space from 1000 to 10bf is equivalent to three cache blocks located at 1000, 1040 and 1080 respectively.

The "loop 8 r1" instruction at imaginary address 1010 specifies that the following 8 processor packets at addresses 1020 to 1090 constitute a loop, and that the loop should be executed a number of times specified by the contents of a register r1. The loop in this example therefore spans all three cache blocks, but neither the start nor the end of the loop is aligned with a cache block boundary.

Fig. 11 shows how the program portion of Fig. 10 is stored in memory after compression. There are three compressed sections VCS1, VCS2 and VCS3. The compressed instructions occupy addresses (real addresses) in the schedule storage unit 12 from 2000 to 206b (again expressed in hexadecimal notation).

Each compressed section VCS has, in its first word, the imaginary address of the first instruction belonging to that section after decompression, i.e. the VPC value on entry to the decompressed cache block produced when the section is decompressed.

The second word of each compressed section VCS contains the decompression key needed to decompress the section concerned. The third and subsequent words of the section contain the non-NOP instructions belonging to the section.

Fig. 12 shows the initial state of the instruction cache 40 and the control registers in the updating unit

148. For the purposes of explanation, it will be assumed that the instruction cache is very small, having just two cache blocks CB0 and CB1. Associated with each cache block is a cache tag CT0 or CT1. Each
5 cache tag CT has the V, CC and IBA fields as described previously with reference to Fig. 7.

In the initial state shown in Fig. 12, i.e. prior to execution of the program portion shown in Fig. 11, both cache blocks CB0 and CB1 are not in use and the V-
10 field of the cache tag associated with each cache block is set to 0. The PC register 50 points to the address 1ffc of the instruction that immediately precedes the Fig. 11 program portion.

When the PC register is incremented to reach 2000
15 the Fig. 11 program portion is entered. In this initial state, as shown in Fig. 12, the VPC register 52 is blank. Accordingly, the instruction fetching unit 46 issues the LOAD signal to the cache loading unit 42 which initiates a cache loading operation to load VCS1
20 into the cache 40. The cache loading unit 42 outputs as the value EVPC the VPC value stored in the first word of the section VCS1. This is needed to initialise the VPC register 52.

Once the VPC register is initialised, the cache
25 block which will be used to store the decompressed instructions of the section VCS1 is reserved.

For the purposes of explanation the VPC values (imaginary addresses) shown in Fig. 11 and used in this example specify the imaginary addresses to a precision
30 of one byte. However, it will be understood that, as each processor packet in this example is 16 bytes (4 instructions each of 4 bytes), to identify a processor packet the 4 least significant bits (lsbs) of the VPC value are not required. Accordingly, in practice the
35 VPC register 52 may not have its 4 lsbs implemented. Also, each cache block contains 64 bytes (4 processor

packets per block) and so to provide an imaginary block address IBA the 6 lsbs of the VPC value are not required. Accordingly, only the higher-order bits of the VPC value down to (and including) the 7th lsb are
5 needed to provide the IBA corresponding to the VPC value. Thus, the IBA corresponding to the VPC value 1000 is 40 (also in hexadecimal notation).

The IBA value is mapped to a unique one of the cache blocks based on a predetermined bit field of the
10 VPC value. In this example, where there are only two cache blocks, the bit field comprises a single bit, which is the 7th lsb of the VPC value. This bit provides the INDEX value used to address a cache tag. When INDEX=0 (even-numbered IBA values) cache tag CT0
15 is addressed, and when INDEX=1 (odd-numbered IBA values) cache tag CT1 is addressed.

In this case, with IBA=40, INDEX=0 and cache tag CT0 is addressed. Its V-field is set to 1 and its IBA field is set to 40, so as to reserve cache block CB0
20 for the instructions of VCS1. The cache loading unit 42 then reads the instructions I1 to I6 and the "loop" instruction contained in VCS1 from addresses 2008 to 2020, decompresses them using the decompression key KEY1 stored at address 2004, and stores the
25 decompressed instructions (including NOP instructions as necessary) in the reserved cache block CB0 at imaginary address 1000 to 103f. The CC value (7), representing the number of non-NOP instructions in the cache block just loaded, is output by the cache loading
30 unit 42 and stored in the CC field of the cache tag CT0. Thus, the compressed section VCS1 located at address 2000 has been loaded into the cache block CB0 at imaginary address 1000.

Now that the cache loading operation is complete
35 the instruction fetching unit issues the FETCH instruction to fetch a processor packet from the

imaginary address 1000 pointed to by the VPC register 50. In this case, as the imaginary address corresponds to an IBA of 40, there is cache hit, and, as a result, the CC register 51 in the updating unit 148 is loaded from the CC field in the matching tag CT0 and the processor packet containing the instructions I1 and I2 is read from the cache block CB0 into the instruction register 54. Accordingly, the instructions I1 and I2 are issued to the execution units in parallel.

The instruction fetching unit 46 then issues the UPDATE signal to the updating unit 148 which increments the VPC register to point to the next processor packet at imaginary address 1010.

After the VPC register has been updated to point to address 1010 the instruction fetching unit 46 issues the FETCH signal again. There is again a cache hit and as a result the processor packet PP containing the "loop" instruction is placed in the instruction register 54, so that the loop instruction is issued. This causes the values in the PC and VPC registers 50 and 52 to be copied to the LPC and LVPC registers 150 and 152 respectively. Before being copied into the LVPC register VPC is incremented to point to the first processor packet after the packet containing the "loop" instruction, i.e. the packet at imaginary address 1020 which contains the instructions I3 and I4.

Furthermore, the loop instruction also results in the IC register 154 being loaded with the value held in the register r1 specified in the loop instruction itself, which is 42 in this example. The number of packets in the loop body, 8 in this example, also specified in the loop instruction itself is loaded into the LSIZE register 156 and a copy of LSIZE is also stored in the LCNT register 158. The resulting state of the instruction cache 40 and the registers in the updating unit 148 is shown in Fig. 14.

At the start of the next instruction cycle the instruction fetching unit 46 fetches a processor packet PP from the imaginary address 1020 pointed to by the VPC register 52. There is a cache hit (cache block CB0 again) and the four instructions, including the instructions I3 and I4, of the processor packet at the imaginary address 1020 are issued in parallel to the execution units. The VPC register 52 is then incremented to point to imaginary address 1030 and the LCNT register 158 is decremented by 1.

In the next instruction cycle the processor packet containing the instructions I5 and I6 is issued. VPC is then incremented to the imaginary address 1040 and LCNT is again decremented by 1 to have the value 6.

In the third cycle of the first iteration of the loop, the instruction fetching unit 46 attempts to fetch a processor packet from imaginary address 1040 which is outside the block of decompressed instructions held in cache block CB0. This is detected because the VPC value of 1040 corresponds to an imaginary block address IBA of 41 which is not contained in the IBA field of any valid cache tag. Thus, the instruction cache 40 responds to the FETCH signal by issuing the MISS signal. In response to the MISS signal the instruction fetching unit 46 issues the LOAD signal, in response to which the updating unit 148 updates the PC register 50 to have the value $PC+4(CC+2)$, where CC is the value held in the CC register 51. Thus, PC now points to the first instruction in the compressed section VCS2 in Fig. 11 at real address 2024. After the PC register 50 has been updated in this way, the cache loading operation is performed by the cache loading unit 42. The resulting state of the instruction cache 40 and the registers in the updating unit 148 is shown in Fig. 15.

As shown in Fig. 15, the compressed section VCS2

is stored, after decompression, in the cache block CB1 (the IBA of 41 makes INDEX=1, which addresses the cache tag CT1) and the associated cache tag CT1 is initialised to have a V-field of 1, a CC field of 6 (there being 6 non-NOP instructions I7 to I12 in VCS2) and an IBA field of 41.

Execution then continues, with the instruction fetching unit issuing processor packets from the imaginary addresses 1040, 1050, 1060 and 1070 and getting cache hits each time. The LCNT register 158 is reduced to the value 2.

When the VPC register 52 reaches 1080 it again strays outside the range of imaginary addresses currently held in the cache and a cache miss occurs. The IBA corresponding to the imaginary address 1080 is 42. As the cache is a directly-mapped cache, the IBA of 42 (INDEX=0) must be mapped to the cache block CB0, with the result that the first block that was loaded (corresponding to the compressed section VCS1) is overwritten with the decompressed instructions of VCS3. The resulting cache state is shown in Fig. 16. The cache tag CT0 associated with cache block CB0 has a V-field of 1, a CC field of 8 (there being 8 non-NOP instructions I13 to I20 in VCS3), and an IBA field of 42.

Processor packets are then fetched in successive instruction cycles from imaginary addresses 1080, 1090 and 10a0 and are issued to the execution units. Each time a packet is fetched the instruction cache outputs as the value ACC the value 8 of the cache tag CT0 associated with the cache block CB0 from which the packet is fetched.

When the processor packet at 10a0 is fetched, the LCNT register reaches 0, indicating the end of the first iteration of the loop. The IC register 154 is decremented by 1. Because it is still greater than 0

the updating unit reloads the PC register 50 from the LPC register 150, reloads the VPC register 52 from the LVPC register 152, and reloads the LCNT register 158 from the LSIZE register 156. The resulting state is shown in Fig. 17.

It can be seen from Fig. 17 that when the instruction fetching unit 46 attempts to fetch a packet from imaginary address 1020, which has a corresponding IBA of 40, there will be a cache miss. Accordingly, after receiving the MISS signal from the instruction cache 40 the instruction fetching unit 46 applies the LOAD signal to the cache loading unit 42 with the result that the compressed section VCS1 at real address 2000 (as pointed to by the PC register 50) is decompressed and reloaded into the cache at cache block CB0. Accordingly, the processor packets having imaginary addresses 1000 to 1030 are again held in the cache block CB0 and the processor packets having imaginary addresses 1040 to 1070 are held in the cache block CB1. The resulting state is shown in Fig. 18.

Execution of instructions continues in this way until all 42 iterations of the loop have been completed. At this point, the IC register 154 is decremented to reach 0. At this time the loop terminates and the instruction fetching unit 46 continues issuing instructions from the processor packet after the last loop-body instruction, i.e. the processor packet containing the instructions I19 and I20 at imaginary address 10b0.

Referring back to the state shown in Fig. 18, it will be assumed that immediately after the processor packet having imaginary address 1020 is issued in the second iteration an interrupt occurs. This interrupt causes the operating system to swap out the current process and begin executing a different process. This may disturb the contents of the cache blocks so that on

return to the original process there is no guarantee that the instructions I1 to I12 belonging to the original process and placed there before the interrupt occurred will still be present in the cache blocks.

5 Accordingly, in the Fig. 8 embodiment when an interrupt occurs the contents of all of the registers of the updating unit 148 are saved by the operating system and are reinstated prior to returning to the original process to resume execution. The content of
10 the instruction cache 40 is not saved.

 Fig. 19 shows the state of the instruction cache and the registers at the point when execution of the original process is resumed at imaginary address 1030. In this example it is assumed that the contents of both
15 cache blocks (corresponding respectively to VCS1 and VCS2) present prior to the interrupt are evicted by the process invoked by the interrupt. For the sake of clarity, the blocks have been shown to be evicted by simply invalidating the associated cache tags and
20 clearing the blocks. In practice, other blocks would be present rather than the cache being empty, but the net effect is the same.

 When the instruction fetching unit 46 attempts to fetch a processor packet from imaginary address 1030 a
25 cache miss will occur. The instruction fetching unit 46 will then issue the LOAD signal to the cache loading unit which loads the compressed section VCS1 pointed to by the restored PC register (pointing to the address 2000). This is the required block of instructions and
30 the resulting state is as shown in Fig. 20.

 As described above, the Fig. 8 embodiment can cope with random and unexpected evictions from the cache even in the presence of simple control transfer operations associated with hardware-controlled basic
35 loops.

 In the embodiments described above, each

compressed section VCS includes the imaginary address for the instructions belonging to that section. However, it will be appreciated that it is not necessary to include such imaginary address information in every one of the compressed sections VCS. For example, the imaginary address information could be omitted from all compressed sections except for the first section of a program to be executed. It is necessary to have imaginary address information in the first section to enable the VPC register to be initialised (cf. Fig. 12 above). However, thereafter the VPC register will always be maintained consistent with the PC register, independently of the VPC values held in the second and subsequent compressed sections of the program.

It may still be advantageous to include the imaginary address information in all compressed sections, or at least in certain compressed sections, for error checking purposes. For example, when a compressed section that includes imaginary address information is loaded into the cache the information included in the section can be compared with the VPC value calculated independently by the updating unit, and an error can be flagged if the information from the compressed section is not consistent with the calculated VPC value.

Fig. 21 shows a flowchart for use in explaining how original instructions (non-compressed instructions) of a program are compressed in one embodiment of the present invention. The compression method is carried out, for example, by an assembler and/or linker of the processor.

In a first step S1, a sequence of original instructions of the program to be compressed (e.g. Fig. 10) is converted into a corresponding sequence of compressed-form instructions (e.g. Fig. 11). For

example, the instructions may be compressed so as to remove therefrom any explicit NOP instructions.

Then, in a step S2 the original instructions are assigned imaginary addresses according to the sequence in which the instructions appeared prior to compression (again see Fig. 10). The assigned imaginary addresses are imaginary addresses at which the instructions are to be considered to exist when held in decompressed form in the instruction cache of the processor.

Finally, in a step S3, the compressed-form instructions are stored in the program memory together with imaginary address information specifying the imaginary addresses assigned in step S2. In this way, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache at execution time, the processor can assign the specified imaginary addresses to the decompressed instructions.

When the compressed-form instructions are stored in the program memory in one or more compressed sections, as described hereinbefore in relation to the first and second embodiments, the compressed-form instructions belonging to each section may occupy one block of the processor's instruction cache when decompressed. In this case, each section may contain imaginary address information relating to the instructions of the section. The imaginary address information may specify, for example, the imaginary address at which a first one of the decompressed instructions corresponding to the compressed section is to be considered to exist when the decompressed instructions are held in the processor's instruction cache.

It will be appreciated that, when assigning the imaginary addresses in step S2, the processor's assembler and/or linker have a responsibility to assign

entry points in the imaginary address space to each compressed section so that, when decompressed, all sections are disjoint in the imaginary address space. The assembler/linker preferably assigns imaginary entry-points that will not create cache conflicts for blocks of decompressed instructions that are likely to be co-resident in the cache. This is not required for correct operation of the processor, but will improve the ratio of cache hits to cache misses at execution time. The entry points in the imaginary address space must all be aligned on processor packet boundaries.

A compression method embodying the present invention can be implemented by a general-purpose computer operating in accordance with a computer program. This computer program may be carried by any suitable carrier medium such as a storage medium (e.g. floppy disk or CD Rom) or a signal. Such a carrier signal could be a signal downloaded via a communications network such as the Internet. The appended computer program claims are to be interpreted as covering a computer program by itself or in any of the above-mentioned forms.

Although the above description relates, by way of example, to a VLIW processor it will be appreciated that the present invention is applicable to processors other than VLIW processors. A processor embodying the present invention may be included as a processor "core" in a highly-integrated "system-on-a-chip" (SOC) for use in multimedia applications, network routers, video mobile phones, intelligent automobiles, digital television, voice recognition, 3D games, etc.

CLAIMS:

1. A processor, for executing instructions of a program stored in compressed form in a program memory, comprising:

5 a program counter for identifying a position in the said program memory;

an instruction cache, having a plurality of cache blocks, each for storing one or more instructions of the said program in decompressed form;

10 cache loading means, including decompression means, operable to perform a cache loading operation in which one or more compressed-form instructions are read from the said position in the program memory identified by the program counter and are decompressed and stored
15 in one of the said cache blocks of the instruction cache;

a cache pointer for identifying a position in the said instruction cache of an instruction to be fetched for execution;

20 instruction fetching means for fetching an instruction to be executed from the position identified by the cache pointer and operable, when a cache miss occurs because the instruction to be fetched is not present in the instruction cache, to cause the cache
25 loading means to perform such a cache loading operation; and

updating means for updating the program counter and cache pointer in response to the fetching of instructions so as to ensure that the said position
30 identified by the said program counter is maintained consistently at the position in the said program memory at which the instruction to be fetched from the instruction cache is stored in compressed form.

35 2. A processor as claimed in claim 1, wherein the said position in the instruction cache of an instruction to be fetched is identified by the said

cache pointer in terms of an imaginary address assigned to the instruction, at which the instruction is considered to exist when held in decompressed form in one of the said cache blocks.

5 3. A processor as claimed in claim 2, wherein the said imaginary address of an instruction is assigned thereto during assembly/linking of the said program based on the sequence of original instructions in the program prior to compression.

10 4. A processor as claimed in claim 2 or 3, wherein imaginary address information, from which the said imaginary address assigned to each instruction is derivable, is stored with the compressed-form instructions in the program memory and is employed in
15 the cache loading operation so as to associate with each decompressed instruction present in the instruction cache the imaginary address assigned thereto.

20 5. A processor as claimed in any preceding claim, wherein:

the compressed-form instructions are stored in the program memory in one or more compressed sections, the compressed-form instructions belonging to each section occupying one of the said cache blocks when
25 decompressed, and at least one section also contains imaginary address information relating to the instructions belonging to the section; and

the said cache loading means are operable, in such a cache loading operation, to decompress and load into
30 one of the said cache blocks one such compressed section stored at the position in the program memory identified by the program counter.

35 6. A processor as claimed in claim 5, wherein the said imaginary address information of said one section specifies the imaginary address at which a first one of the decompressed instructions

corresponding to the compressed section is considered to exist when the decompressed instructions are held in one of the cache blocks.

5 7. A processor as claimed in claim 5 or 6, wherein in the said cache loading operation the cache block into which the decompressed instructions of the compressed section are loaded is assigned an imaginary block address based on the said imaginary address assigned to an instruction contained in the section.

10 8. A processor as claimed in claim 7, wherein each said cache block has an associated cache tag in which is stored the said imaginary block address assigned to the cache block with which the cache tag is associated.

15 9. A processor as claimed in any one of claims 5 to 8, wherein the said imaginary address information is contained in only a first one of the said compressed sections to be loaded.

20 10. A processor as claimed in any one of claims 5 to 8, wherein each said compressed section contains imaginary address information relating to the instructions belonging to the section concerned.

25 11. A processor as claimed in any one of claims 5 to 10, wherein the or each said compressed section further contains a decompression key which is employed by the said decompression means to effect the decompression of the instructions belonging to the section during the cache loading operation.

30 12. A processor as claimed in claim 11, wherein the instructions of the said program include, prior to compression, preselected instructions that are not stored explicitly in any said compressed section, and the decompression key of the or each compressed section identifies the positions at which the preselected
35 instructions are to appear in the cache block when the compressed section is decompressed.

13. A processor as claimed in claim 12, wherein the said preselected instructions are "no operation" instructions.

5 14. A processor as claimed in any one of claims 5 to 13, wherein the said updating means include:

next-section locating means operable, in the event of such a cache miss, to employ next-section-locating information, stored in association with the cache block which has accessed most recently to fetch an
10 instruction, to locate the position in the program memory of a next compressed section following the compressed section corresponding to that most-recently-accessed cache block.

15 15. A processor as claimed in claim 14, wherein such next-section-locating information is stored in association with each cache block in which valid decompressed instructions are held, the stored information being for use in locating the position in the program memory of the next compressed section
20 following the compressed section corresponding to the cache block concerned.

16. A processor as claimed in claim 15, wherein the next-section-locating information is stored in association with each cache block when that block is
25 loaded in such a cache loading operation.

17. A processor as claimed in any one of claims 14 to 16, wherein the said next-section-locating information associated with the cache block relates to a size of the compressed section corresponding to that
30 cache block.

18. A processor as claimed in claim 17, wherein the said size is determined by the cache loading means when loading the cache block in the cache loading operation.

35 19. A processor as claimed in claim 15, wherein the updating means include:

locating information register means for storing the said next-section-locating information associated with the most-recently-accessed cache block; and

5 copying means operable, when an instruction held in one of the cache blocks is fetched, to copy into the locating information register means the said next-section-locating information stored in association with that block;

10 the said next-section-locating means being operable, in the event of such a cache miss, to employ the next-section-locating information stored in the location information register means to locate the said position of the said next compressed section.

15 20. A processor as claimed in any one of claims 14 to 19 when read as appended to claim 12 or 13, wherein:

20 the said next-section-locating information associated with the cache block represents the number of instructions held in that cache block that are not said preselected instructions.

25 21. A processor as claimed in claim 20, wherein the decompression means include counter means operable, during such a cache loading operation, to count the number of decompressed instructions that are not said preselected instructions.

22. A processor as claimed in any preceding claim, operable to execute a hardware-controlled loop, wherein:

30 the said updating means further include respective first and second loop control registers and are operable, upon initiation of execution of such a hardware-controlled loop, to cause the program-counter value to be stored in the said first loop control register and to cause the cache-pointer value to be
35 stored in the said second loop control register, and further operable, upon commencement of each iteration

of the loop after the said first iteration thereof, to reload the said program counter with the value held in the said first loop control register and to reload the said cache pointer with the value held in the said
5 second loop control register.

23. A processor as claimed in any preceding claim, wherein the instructions of the said program include very-long-instruction-word (VLIW) instructions.

24. A processor as claimed in any preceding
10 claim, wherein the updating means are operable, when an interrupt occurs during execution of a program, to cause the program-counter value and cache-pointer value to be saved pending handling of the interrupt, and, when the execution of the program is resumed, to cause
15 the saved values to be restored in the program counter and cache pointer.

25. A processor as claimed in any one of claims 14 to 21, wherein the updating means are operable, when an interrupt occurs during execution of a program, to
20 cause the said next-section locating information associated with the most-recently-accessed cache block to be saved pending handling of the interrupt, and, when execution of the program is resumed, to cause the saved next-section locating information to be restored.

26. A processor as claimed in claim 20, wherein
25 the updating means are operable, when an interrupt occurs during execution of a program, to cause the values held in the said loop control registers to be saved pending handling of the interrupt, and, when
30 execution of the program is resumed, to cause the saved values to be restored in the loop control registers.

27. A method of compressing a program to be
executed by a processor in which compressed-form
instructions stored in a program memory are
35 decompressed and cached in an instruction cache prior to being issued, the method including the steps of:

converting a sequence of original instructions of the program into a corresponding sequence of such compressed-form instructions;

5 assigning such original instructions imaginary addresses according to the said sequence thereof, the assigned imaginary addresses being imaginary addresses at which the instructions are to be considered to exist when held in decompressed form in the said instruction cache of the processor; and

10 storing, in the said program memory, the compressed-form instructions together with imaginary address information specifying the said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the
15 processor into the instruction cache, the processor can assign the specified imaginary addresses to the decompressed instructions.

20 28. A method as claimed in claim 27, wherein the assigned imaginary addresses are selected so that instructions likely to coexist in the instruction cache at execution time will not be mapped to the same cache block.

25 29. A method as claimed in claim 27 or 28, wherein the compressed-form instructions are stored in the said program memory in one or more compressed sections, the compressed-form instructions belonging to each section occupying one cache block of the processor's instruction cache when decompressed, and at
30 least one compressed section also containing imaginary address information relating to the instructions of that section.

35 30. A method as claimed in claim 29, wherein the said imaginary address information specifies the imaginary address at which a first one of the decompressed instructions corresponding to the said one compressed section is to be considered to exist when

the decompressed instructions are held in the said instruction cache.

31. A method as claimed in claim 29 or 30, wherein the said imaginary address information is
5 contained in only a first one of the said compressed sections to be loaded.

32. A method as claimed in claim 29 or 30, wherein each said compressed section contains imaginary address information relating to the instructions
10 belonging to the section concerned.

33. A method as claimed in any one of claims 29 to 32, wherein the or each said compressed section further contains a decompression key for use by the processor to carry out the decompression of the
15 instructions belonging to the said section.

34. A method as claimed in claim 33, wherein the said sequence of original instructions of the program includes preselected instructions that are not stored explicitly in any said compressed section, and the
20 decompression key of the or each said compressed section identifies the positions at which said preselected instructions exist are to appear in a decompressed sequence of instructions corresponding to the section.

25 35. A method as claimed in claim 34, wherein the said preselected instructions are "no operation" instructions.

36. A computer program which, when run on a computer, causes the computer to carry out the
30 compression method of any one of claims 27 to 35.

37. A computer program which, when run on a computer, causes the computer to carry out a method of compressing a processor program to be executed by a processor, the processor being operable to decompress
35 compressed-form instructions stored in a program memory and to cache the decompressed instructions in an

instruction cache prior to issuing them, the computer program including:

5 a converting portion for converting a sequence of original instructions of the processor program into a corresponding sequence of such compressed-form instructions;

10 an assigning portion for assigning such original instructions imaginary addresses according to the said sequence thereof, the assigned imaginary addresses being imaginary address at which the instructions are to be considered to exist when held in decompressed form in the said instruction cache of the processor; and

15 a storing portion for storing, in the said program memory, the compressed-form instructions together with imaginary address information specifying the said assigned imaginary addresses so that, when the compressed-form instructions are decompressed and loaded by the processor into the instruction cache, the
20 processor can assign the specified imaginary addresses to the decompressed instructions.

38. A computer program as claimed in claim 36 or 37; carried by a carrier medium.

25 39. A computer program as claimed in claim 38, wherein the said carrier medium is a storage medium.

40. A computer program as claimed in claim 38, wherein the said carrier medium is a signal.

41. A processor substantially as hereinbefore described with reference to the accompanying drawings.

30 42. A method for compressing a program substantially as hereinbefore described with reference to the accompanying drawings.

35 43. A computer program which, when run on a computer, causes the computer to carry out a compression method substantially as hereinbefore described with reference to the accompanying drawings.

ABSTRACT

PROCESSORS HAVING COMPRESSED INSTRUCTIONS AND
METHODS OF COMPRESSING INSTRUCTIONS FOR PROCESSORS

5 Instructions of a program are stored in compressed
form in a program memory (12). In a processor which
executes the instructions, a program counter (50)
identifies a position in the program memory. An
instruction cache (40) has cache blocks, each for
10 storing one or more instructions of the program in
decompressed form. A cache loading unit (42) includes
a decompression section (44) and performs a cache
loading operation in which one or more compressed-form
instructions are read from the position in the program
15 memory identified by the program counter and are
decompressed and stored in one of the said cache blocks
of the instruction cache. A cache pointer (52)
identifies a position in the instruction cache of an
instruction to be fetched for execution. An
20 instruction fetching unit (46) fetches an instruction
to be executed from the position identified by the
cache pointer. When a cache miss occurs because the
instruction to be fetched is not present in the
instruction cache, the cache loading means performs
25 such a cache loading operation. An updating unit (48)
updates the program counter and cache pointer in
response to the fetching of instructions so as to
ensure that the position identified by the said program
counter is maintained consistently at the position in
30 the program memory at which the instruction to be
fetched from the instruction cache is stored in
compressed form.

35 [Fig. 3]

1/13

	instruction 1	instruction 2	
0	add	nop	← P0
2	nop	mpy	← P1
4	nop	load	~ P2

FIG. 1(A)

before compaction

0	add
1	nop
2	nop
3	mpy
4	nop
5	load

FIG. 1(B)

after compaction

0	100101	key
1	add	
2	mpy	
3	load	

FIG. 1(C)

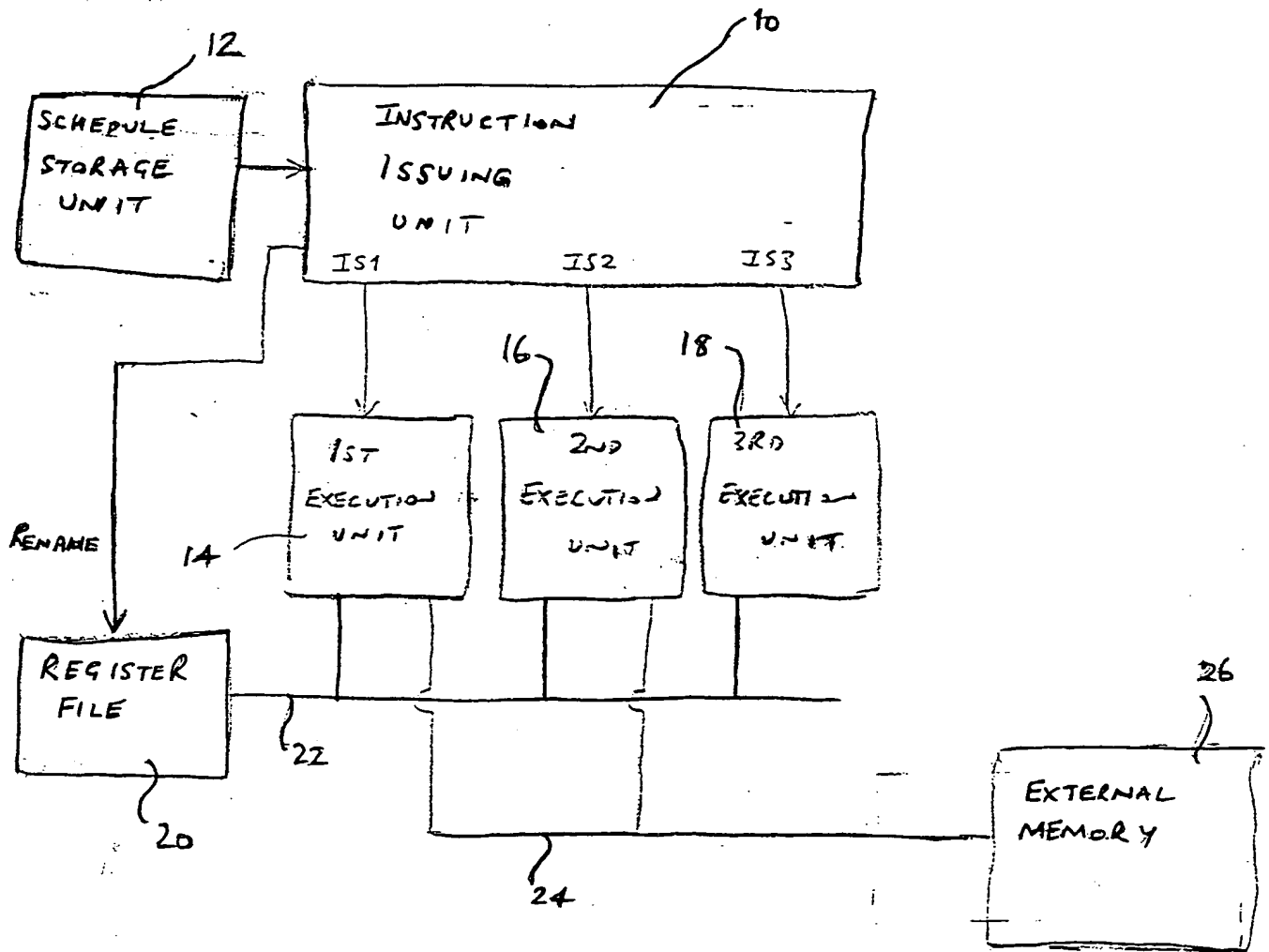


FIGURE 2

10

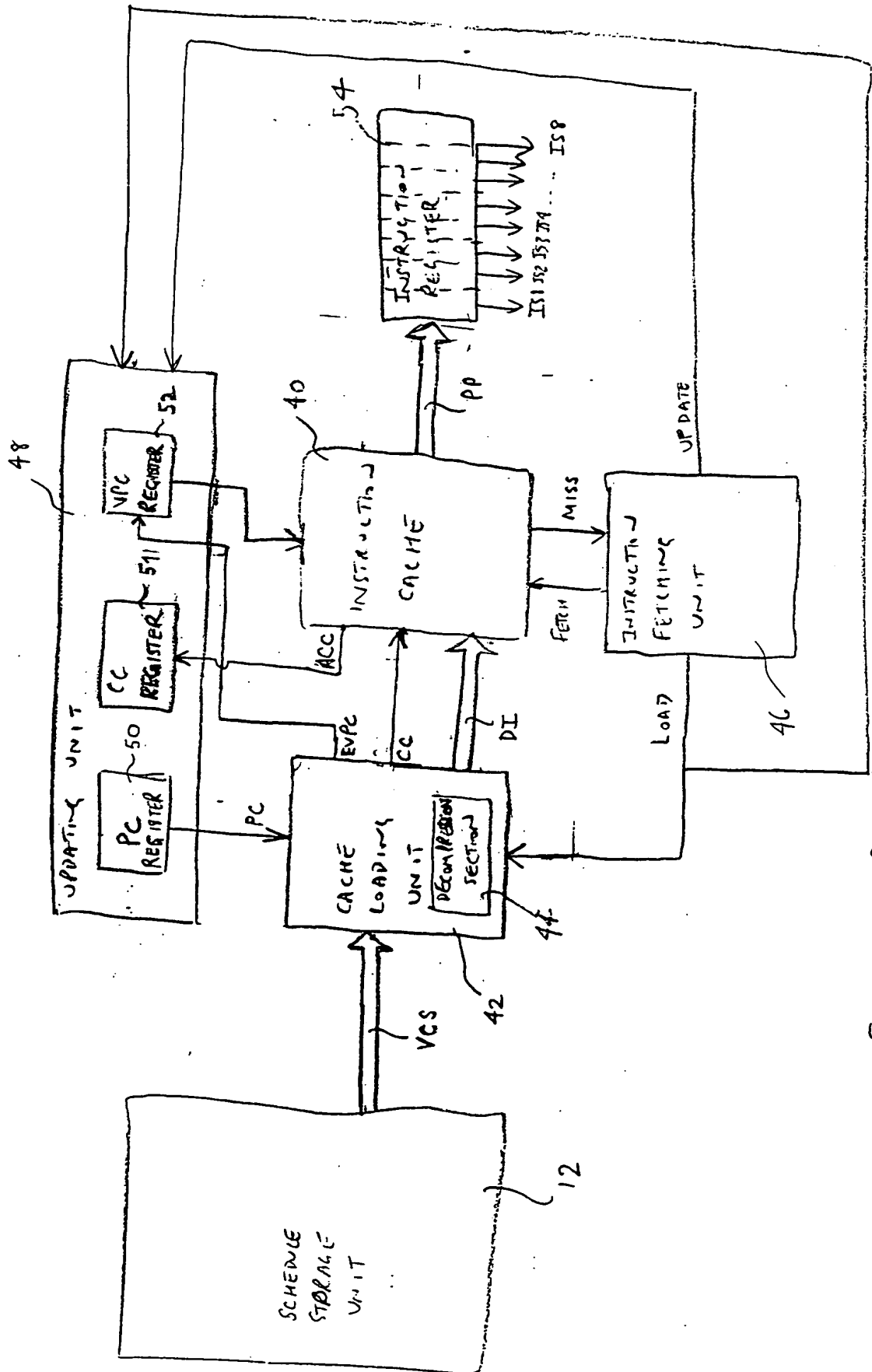


FIGURE 3

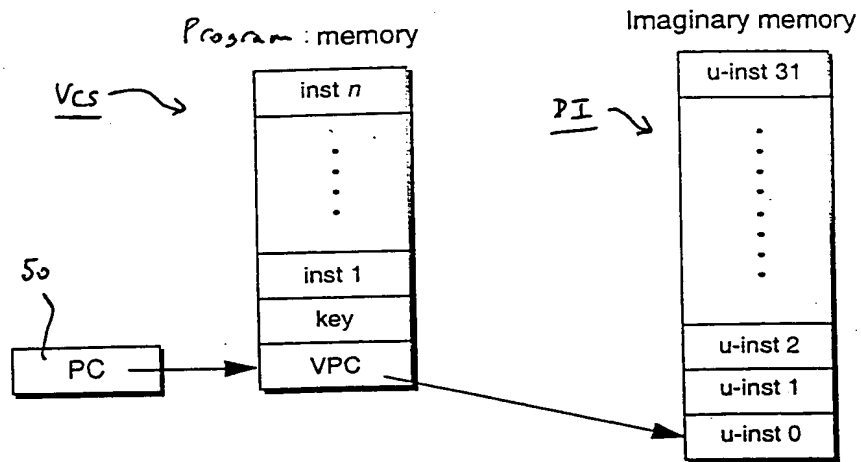


Fig. 4

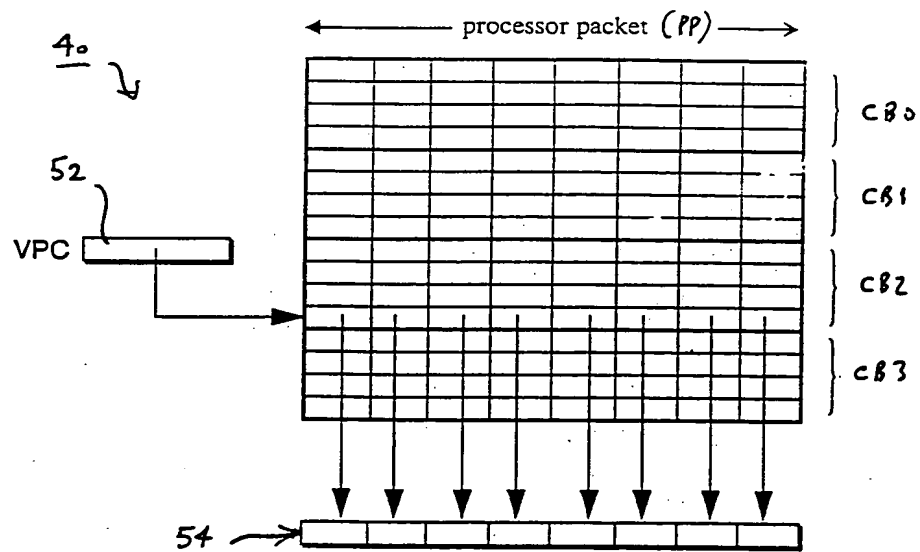


Fig. 5

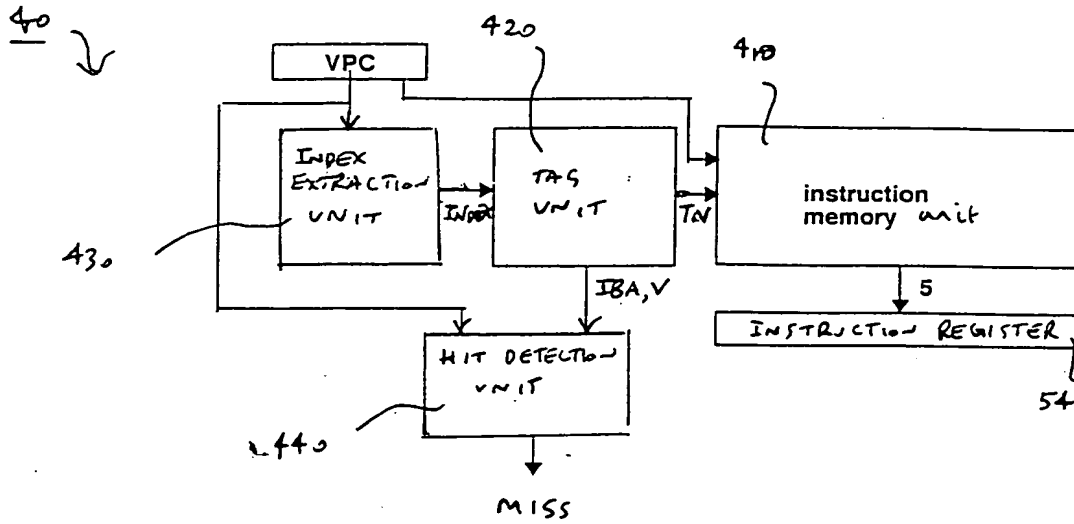


FIG. 6

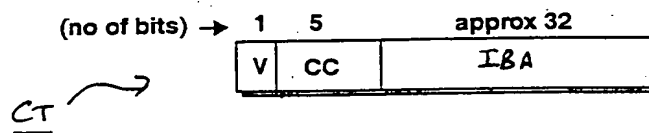


FIG. 7

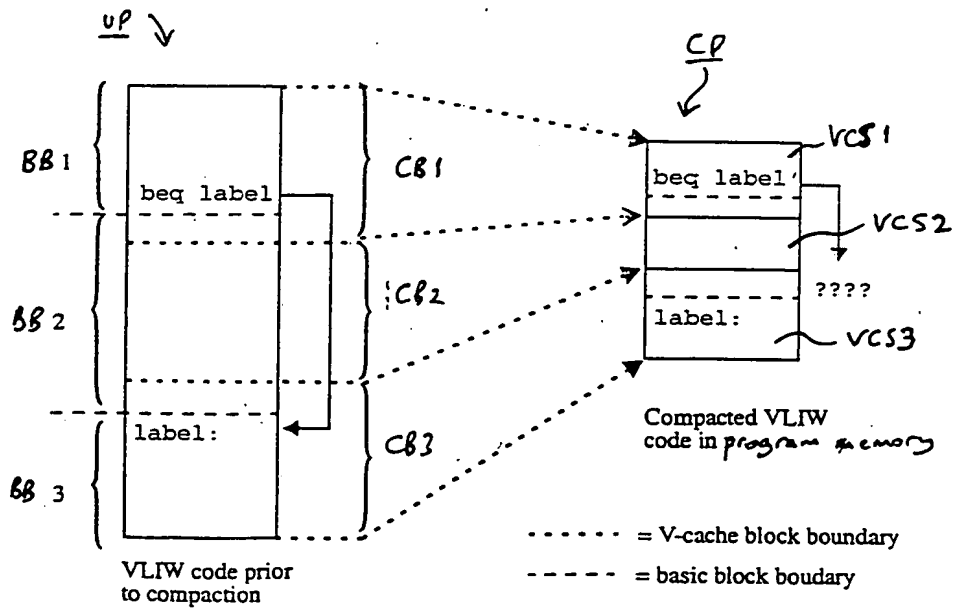


FIG. 9

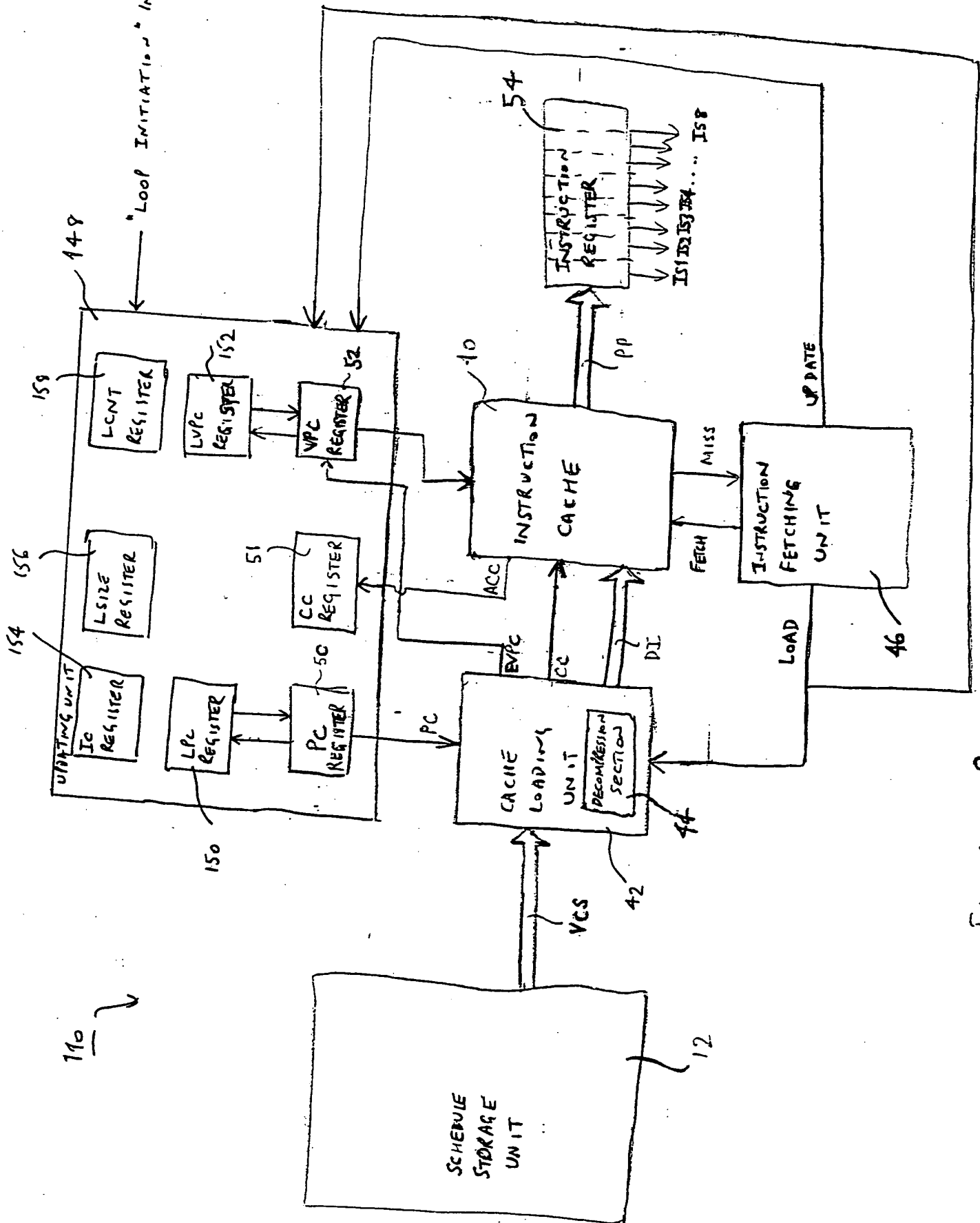


FIGURE 8

-	I1	I2	-	1000
loop 8, r1	-	-	-	1010
I3	I4	-	-	1020
-	I5	-	I6	1030
-	I7	-	-	1040
I8	I9	-	-	1050
-	I10	-	I11	1060
-	-	-	I12	1070
-	I13	I14	-	1080
I15	-	I16	-	1090
-	I17	-	I18	10a0
I19	-	-	I20	10b0

loop:

Fig. 10

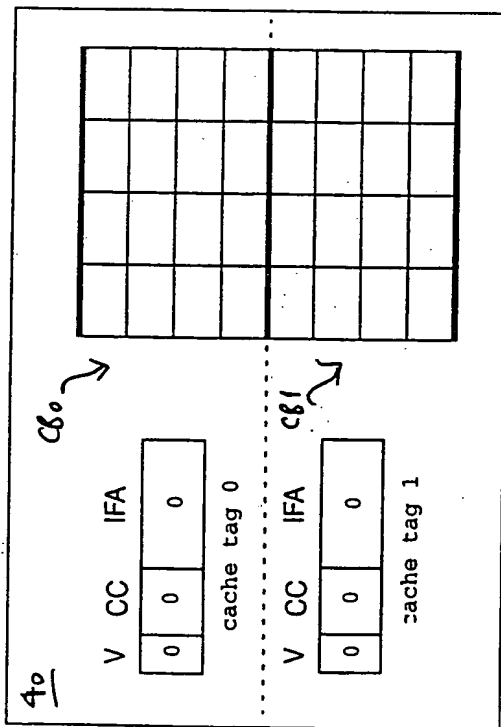
1000	2000
key1	2004
I1	2008
I2	200c
loop 8, r1	2010
I3	2014
I4	2018
I5	201c
I6	2020
1040	2024
key2	2028
I7	202c
I8	2030
I9	2034
I10	2038
I11	203c
I12	2040
1080	2044
key3	2048
I13	204c
I14	2050
I15	2054
I16	2058
I17	205c
I18	2060
I19	2064
I20	2068

VCS1

VCS2

VCS3

Fig. 11



148

PC Iffc

VPC

LVPC

CC

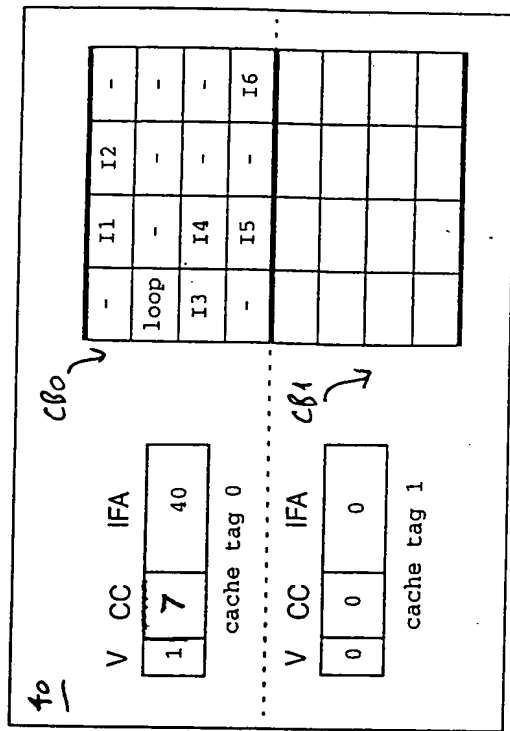
IC

LSIZE

LCNT

LPC

FIG. 12



148

PC 2000

VPC 1000

LVPC

CC 7

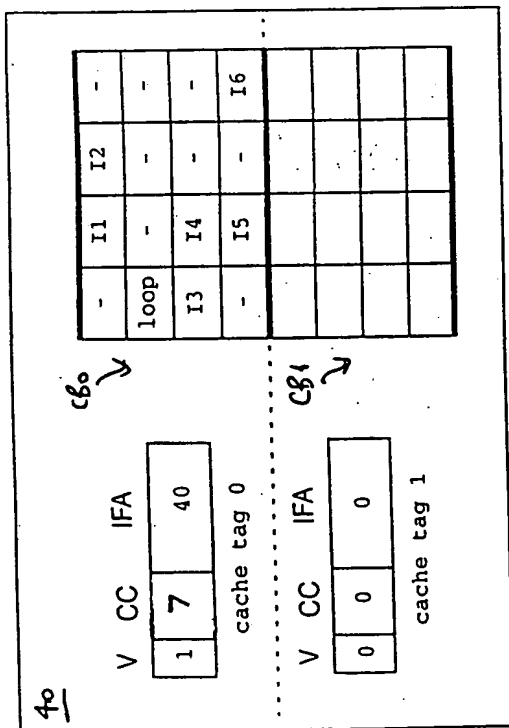
IC

LSIZE

LCNT

LPC

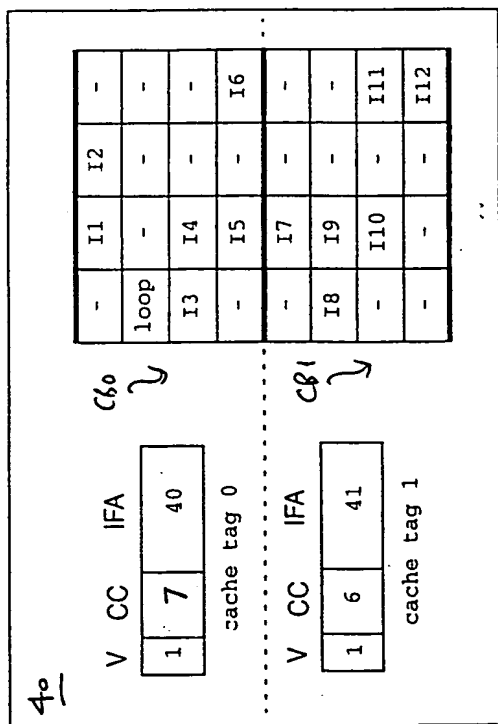
FIG. 13



148

PC	2000
VPC	1020
LVPC	1020
CC	7
IC	42
LSIZE	8
LCNT	8
LPC	2000

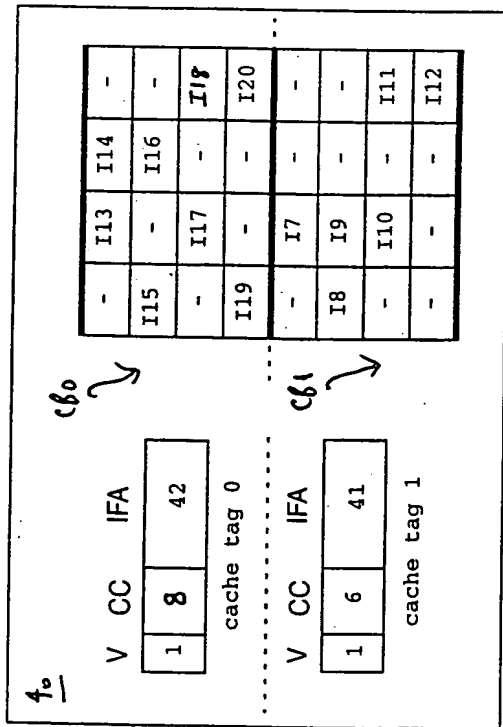
Fig. 14



148

PC	2024
VPC	1040
LVPC	1020
CC	7
IC	42
LSIZE	8
LCNT	6
LPC	2000

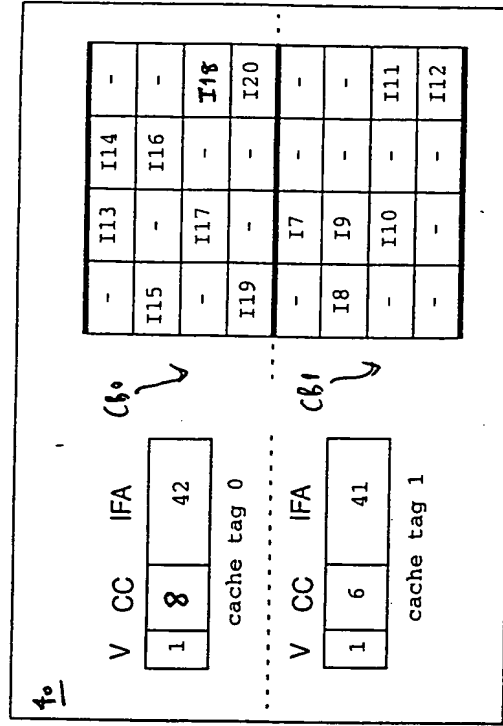
Fig. 15



148 148

PC	2044
VPC	1080
LVPC	1020
CC	8
IC	42
LSIZE	8
LCNT	2
LPC	2000

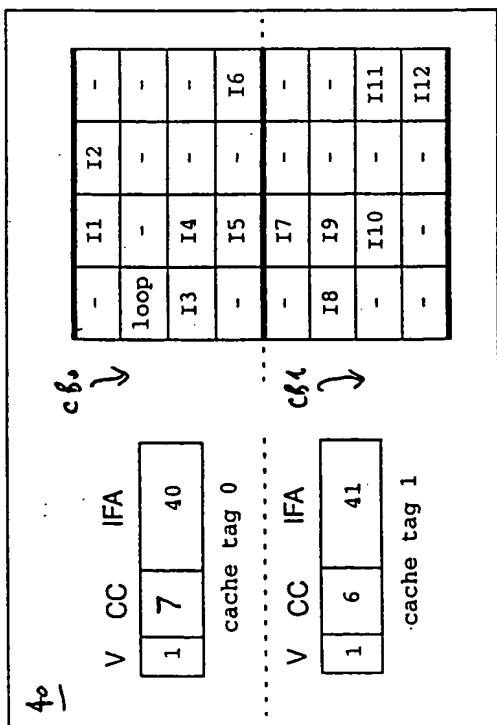
Fig. 16



148 148

PC	2000
VPC	1020
LVPC	1020
CC	8
IC	41
LSIZE	8
LCNT	8
LPC	2000

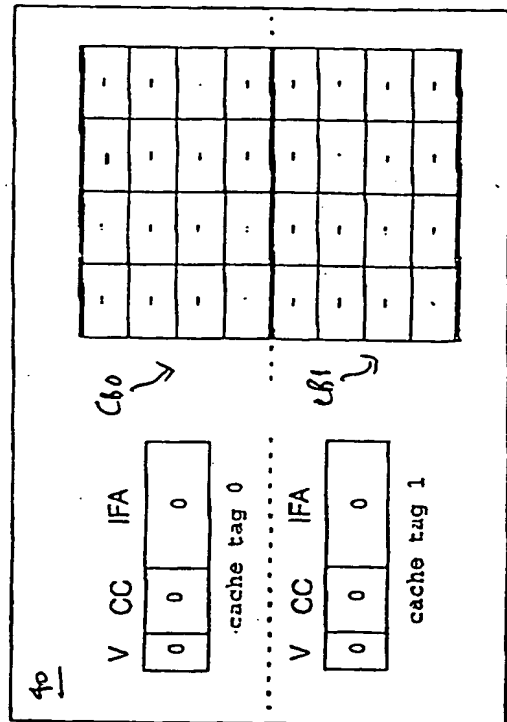
Fig. 17



44g

PC	2000
VPC	1020
LVPC	1020
CC	7
IC	41
LSIZE	8
LCNT	8
LPC	2000

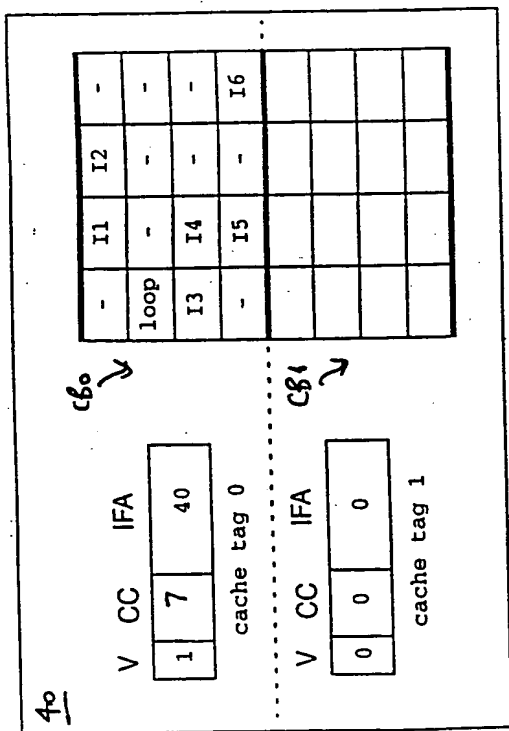
Fig. 18



44g

PC	2000
VPC	1020
LVPC	1020
CC	7
IC	41
LSIZE	8
LCNT	8
LPC	2000

Fig. 19



148

PC	2000
VPC	1020
LVPC	1020
CC	7
IC	41
LSIZE	8
LCNT	8
LPC	2000

Fig. 20

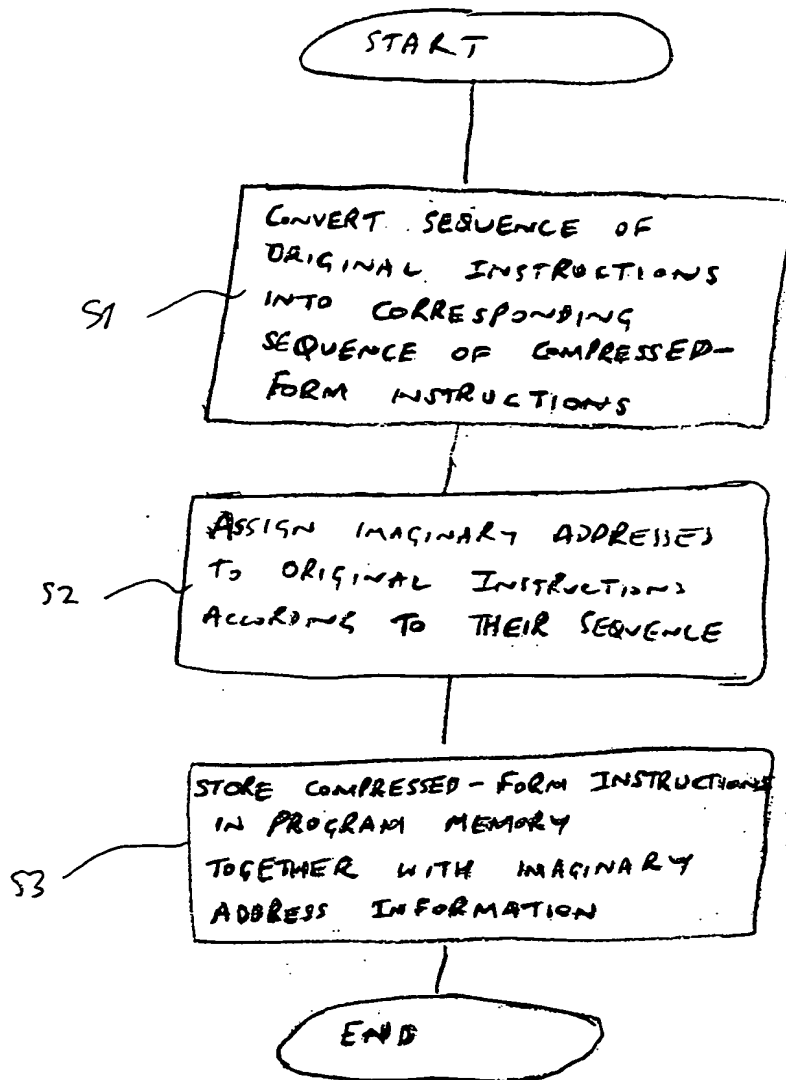


FIG. 21

THIS PAGE BLANK (USPTO)